View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by Edinburgh Research Explorer

THE UNIVERSITY of EDINBURGH

Edinburgh Research Explorer

A Formal Framework for Provenance Security

Citation for published version:
Cheney, J 2011, A Formal Framework for Provenance Security. in Proceedings of the 2011 IEEE 24th Computer Security Foundations Symposium. IEEE Computer Society Press, Washington, DC, USA, pp. 281-293. DOI: 10.1109/CSF.2011.26

Digital Object Identifier (DOI):
10.1109/CSF.2011.26

Link:
Link to publication record in Edinburgh Research Explorer

Document Version:
Peer reviewed version

Published In:
Proceedings of the 2011 IEEE 24th Computer Security Foundations Symposium

General rights
Copyright for the publications made accessible via the Edinburgh Research Explorer is retained by the author(s) and / or other copyright owners and it is a condition of accessing these publications that users recognise and abide by the legal requirements associated with these rights.

Take down policy
The University of Edinburgh has made every reasonable effort to ensure that Edinburgh Research Explorer content complies with UK legislation. If you believe that the public display of this file breaches copyright please contact openaccess@ed.ac.uk providing details, and we will remove access to the work immediately and investigate your claim.

Download date: 05. Apr. 2019

A formal framework for provenance security

James Cheney
Laboratory for Foundations of Computer Science
University of Edinburgh
Edinburgh, Scotland
Email: jcheney@inf.ed.ac.uk

***Abstract*—Provenance, or information about the origin, derivation, or history of data, is becoming an important topic especially for shared scientific or public data on the Web. It clearly has implications on security (and vice versa) yet these implications are not well-understood. A great deal of work has focused on mechanisms for recording, managing or using some kind of provenance information, but relatively little progress has been made on foundational models that define provenance and relate it to security goals such as availability, confidentiality or privacy. We argue that such foundations are essential to making meaningful progress on these problems and should be developed. In this paper, we outline a formal model of provenance, propose formalizations of security properties for provenance such as *disclosure* and *obfuscation*, and explore their implications in domains based on automata, database queries and workflow provenance graphs.**

***Keywords*-provenance; semantics; security**

I. Introduction

Provenance is, informally, information about the history, derivation, origin, or context of an artifact. In computational settings, provenance is data that describes some other data, or the computation or process that produced it. Absence of provenance, or incorrect provenance, can lead to a new class of failures in computer systems, which we call *provenance failures*. For example:

- Absence of date information on news articles has led to old news being reused by aggregators such as Google News. This in turn can lead to economic losses if investors misinterpret this information, for example when the old news is about a company's near-bankruptcy [1]. Even simple forms of provenance such as "when was this data created" are not currently maintained effectively on the Web.
- Lack of transparency or information needed to repeat scientific computations makes it difficult for reviewers to evaluate scientific contributions and for authors to avoid unintentional error. Bugs in programs or scripts have already led to publication of articles with hard-to-find errors that ultimately had to be retracted [2]. Many provenance techniques are aimed at aiding repeatability or transparency in scientific computation.
- Privacy or confidentiality can be violated if provenance information is unintentionally made available, for example, if government documents are accidentally published as Word documents with embedded change history [3].

A number of researchers and organizations, including the W3C, have suggested developing a *provenance infrastructure* or systems and standards that make it possible to determine the provenance of data on the Web [4], [5]. Further details and examples of provenance failures, and our views on the need for further research on these issues, are summarized in the paper [6].

The question of how to make these informal motivations formal, or to precisely characterize what makes some *mechanism* for provenance suitable for enforcing a desired *policy*, has largely been ignored in the literature on provenance. A great deal of work has focused on design of efficient techniques for recording some additional so-called provenance information, often in the context of a specific system (see e.g. [7]–[11]); other work has emphasized defining formal models of provenance and exploring their properties [12], [13], or identifying and solving particular decision problems that arise for forms of data frequently used as provenance, such as directed acyclic graphs [14], [15]. However, the wide variety of such models and the applications that have inspired them begs several questions:

1) What is (and isn't) provenance?
2) Are there natural choices for the "best" or most informative provenance for a given system?
3) How do we know whether a system records correct provenance, or records enough provenance for a given application?
4) How can we compare provenance techniques in terms of expressiveness or generality?
5) What kinds of security properties (availability, confidentiality, privacy, nonrepudiation) should provenance have, and what kinds of policies can be defined and effectively enforced?

It is important to note that these are really questions about the *semantics* of provenance, and they will only become more important as provenance technology becomes deployed on the Web or other distributed settings. Among these, the first three seem to address subjective concerns where there is not necessarily a unique way of framing the problem or deriving solutions. The many different models for and

implementations of provenance in different systems, such as databases [12], [13], workflow management systems [7], [8], and storage systems [10], [11], suggests that there may be no single definition that applies to all these situations, nor an obvious choice of "best" or most general provenance mechanism even for a specific system. Indeed, it seems likely that the answer to question (2) is actually "no", since any real system can be modeled at finer and finer levels of granularity.

Concerning the third question, most work on particular systems seems to assume that it is obvious that some information can be interpreted usefully as provenance, but when data and provenance crosses systems boundaries this implicit knowledge may be lost, and there may be many new avenues for corrupting provenance records through errors, inconsistencies or attacks. Moreover, as the fourth question highlights, when different approaches to provenance are investigated, it is important to be able to relate both their performance and their expressiveness or generality, to avoid apples-to-oranges comparisons.

Finally, as raised by the fifth question, these concerns become especially pressing when we wish to consider the interaction between provenance and security. The implications of provenance for security (and vice versa) have only recently begun to be investigated [16]–[20]. Obviously, provenance information may need to be protected just like any other data produced or consumed by computer systems. Less obviously, as discussed by Braun et al. [17] provenance may have subtle interactions with confidentiality of ordinary data (e.g. provenance may permit new inferences leading to information leakage) or with privacy (e.g. there may be a tension between making provenance information available and protecting the identities of principals contributing to computations). These situations are especially complex when provenance crosses system boundaries or domains of control involving principals with different goals and needs, or for stateful systems such as scientific databases, which grow or change over time as a result of contributions by many contributors [21].

Most work on provenance security so far has essentially reapplied known mechanisms such as access control, digital signatures, information flow control, or privacy without seeking to answer the above semantic questions. One important exception is Chong's proposals of definitions of provenance security policies [22], drawing on the trace model introduced by Acar, Ahmed and Cheney [23]. Some work on provenance has also been inspired by ideas in security, such as dependency provenance [24], which provides a noninterference-like criterion for ensuring that all possible dependencies of a part of the output are tracked. Davidson et al. [25] propose adapting ideas from database privacy and anonymity and give candidate formal definitions of privacy policies for workflow provenance. Nevertheless, these only represent first steps.

Addressing these questions properly requires a semantic framework that makes some reasonable assumptions about the subjective parts of the picture. In this paper, we develop such a framework, and propose some general semantic definitions of security properties for provenance. The main contribution of this paper is the framework and definitions themselves, but we justify our view that the framework is general and useful by exploring its ramifications using example domains including finite automata and transducers, database queries, and simple workflow-like programs.

We assume that a number of principals interact with some system equipped with a set of possible behaviors, called *traces*. Traces are expected to contain all information about the system behavior in which any principal is potentially interested. This is a subjective judgement, and so the design of the language of traces needs to be agreed by the principals or system designers. Thus, the trace forms a "most general" form of provenance for the system, at least as far as the principals interacting with it are concerned. However, different kinds of traces might be suitable for different applications, and we do not prescribe a strategy for designing or implementing provenance tracking mechanisms. Much research on provenance focuses on exactly these problems. We also do not assume that there is necessarily a practical strategy for recording the traces. They are only an abstraction used for relating other more specific forms of provenance, including the ones of practical interest.

In principle, one could make the complete trace available to each principal authorized to use the system. However, this default strategy might be impractical or undesirable. For example, it is typically not practical to record traces at the level of individual instructions in production systems, particularly scientific computations where performance is paramount. Moreover, the principals may have different privileges concerning what aspects of system behavior they are allowed to observe. These observations may permit principals to draw inferences about aspects of the system that they are not allowed to observe directly, leading to security vulnerabilities.

In the absence of security concerns, provenance is somewhat loosely specified: we can simply try to make as much trace information available as is practical, and let users make whatever sense of it they can. Likewise, if security is the only concern, then provenance is also loosely specified: we can protect the system best by not making any additional provenance information available, and using conventional techniques for security. However, if some knowledge about system behavior needs to be protected while other information is disclosed, there is a basic tension between provenance and security, which further constrains the problem. The main contribution of this paper is to explain what it means for a provenance tracking system to successfully *disclose* some information that users require while *obfuscating* other sensitive information.

To make these constraints precise, we introduce the notion of a *provenance query* Q, which is simply a set of traces that share some property that needs to be obfuscated or disclosed. Furthermore, we consider spaces of *provenance views* P_A which are essentially functions on traces that hide some of the trace information, describing what information is made available by the system to principal A.

Using these ingredients we define the notion of a provenance framework and of a system that is an instance of the framework. We can consider *provenance policies* including:

- Disclosure: Ensuring that a given principal A can always determine whether the actual trace satisfies a given provenance query Q. (That is, ensuring Q is always answerable using P_A.)
- Obfuscation: Ensuring that a given principal A can never be certain whether the actual trace satisfies a given provenance query Q. (That is, ensuring that Q can never be answered by observing P_A.)

Moreover, we further focus attention on *trace-invariant* queries and policies that are independent of the provenance information itself, such as queries about the input or source program used to derive a given output. We ultimately want to understand a number of different classes of trace-invariant policies:

- *availability*: how can we ensure that some information about the input is disclosed to a principal?
- *confidentiality*: how can we ensure that confidential information is never indirectly disclosed to a principal? This is essentially what is studied by Chong's "data security" [22].
- *integrity*: how can provenance records increase trust in the integrity of a process that created some data, and how can such records be protected from corruption? This is a motivation for work on provenance in *curated databases* [26] and security aspects of this situation are also considered by Zhang et al. [18] and Hasan et al. [10], [16].
- *reverse-engineering*: how much information does the provenance (and possibly input and output) provide about the program that produced the results? This is a motivation for work on workflow provenance security [25].
- *explanation*: how much information does the provenance reveal about the possible result values in hypothetical, counterfactual situations? This is part of the motivation for dependency provenance [24] and provenance traces [23], and is an implicit motivation for viewing provenance graphs as conveying causal information [14], [27].

We develop specific instances of our framework with these provenance queries in mind, focusing on the availability and confidentiality properties. We model a principal's knowledge by sets of possible worlds (i.e., sets of traces) and measure the knowledge gain in terms of the worlds considered possible by each principal after observing the provenance. In particular our examples show that provenance can enforce trace-invariant policies that cannot be enforced otherwise.

Although the examples discussed earlier have largely drawn on motivations for provenance on the Web for inspiration, provenance is also widely studied in several other settings, including workflows, databases, and storage systems. In the long term, as these systems become more tightly integrated over the Web, we will need provenance models that transcend the details of these different models. In this paper, we seek to abstract from the specific details of these systems in order to gain an understanding of the common security issues that arise in any system that tracks provenance. Our complexity results show that these definitions play out in subtly different ways in different settings, including automata traces, workflow runs and annotated databases, leading to different answers to questions of decidability and complexity for provenance. One could argue that this suggests that our trace formalism is too general and flexible to model provenance well; however, even if this is the case, this kind of exploration is a necessary first step towards identifying the right level of abstraction.

We also draw a distinction between *static* provenance that describes a single run or behavior of a system, versus *dynamic* provenance that describes a sequence of runs or concurrent interactions between the system, principals and possibly other systems. In this paper, we focus on developing a unifying model and identifying security properties for the easier (but still nontrivial) static case. In fact, most work on provenance in extant database and workflow systems focuses on the static case, so it is reasonable to focus on this case. We leave the development of a formal model for dynamic provenance security for future work.

The structure of the rest of this paper is as follows. In Section II we describe our general model, including the idea of provenance frameworks, systems, and definitions of the disclosure and obfuscation properties, as well as trace-invariant special cases. In Section III we introduce the three main examples we will use to illustrate the model. In Section IV we present the main technical results, which explore the ramifications of the model using the example settings. In Section V we discuss related work and in Section VI we summarize and discuss some directions for future work.

II. Provenance framework

In this section we present our model. We assume that a system designer or community agrees on a set of traces $\mathcal{T}$ of possible executions of the system. These traces constitute a record of system behavior that is assumed to be sufficiently informative for all principals involved. This is an assumption of the model, not a theorem, and we do not further prescribe

design criteria for the space of traces or means of associating them with computations. We believe these decisions ultimately need to be informed both by domain knowledge and principles of clean specifications.

A *provenance query* is a set of traces, or equivalently a function from $\mathcal{T} \to \mathbb{B}$ where $\mathbb{B} = \{0, 1\}$. A *provenance interpretation* is a function $P : \mathcal{T} \to \Omega$ where Ω is some set of *observations* associated with P.

A *provenance framework* $(\mathcal{T}, \mathcal{T}_0, \mathcal{Q}, \mathcal{O}, \mathcal{P})$ consists of a set of traces $\mathcal{T}$, a set of *realizable* traces $\mathcal{T}_0$, a collection $\mathcal{Q}$ of subsets of $\mathcal{T}$ called *trace queries*, a collection $\mathcal{O}$ of observation sets Ω, along with a collection of transformers $h : \Omega \to \Omega'$ that includes identity maps and is closed under composition, and $\mathcal{P}$ is a set of *provenance views* $P : \mathcal{T} \to \Omega$ that is closed under composition with observation transformers. We also assume that $\mathcal{O}$ contains a singleton set $\{\star\}$ and is closed under Cartesian products. [1]

For example, given a set $\mathcal{T}$ of traces we can construct a default provenance framework where $\mathcal{T}_0 = \mathcal{T}$, the queries are all subsets of $\mathcal{T}$, and the objects of $\mathcal{P}$ are all functions of the form $P : \mathcal{T} \to X$ where X is any set, and the arrows are all possible functions over sets. However, the queries and observables might be more restricted, for example in the framework **AUT** we will consider later, $\mathcal{T}$ is a set of strings and $\mathcal{T}_0$ is the language of traces of a given automaton; queries are regular languages; and the category of observations has regular languages as objects and transformations definable by finite transducers as arrows.

We assume that principals know the possible traces but do not have direct access to the trace t realized in a particular scenario. Instead, a principal A can make certain observations of the trace, given by a fixed provenance interpretation $P_A : \mathcal{T} \to \Omega_A$, mapping traces to elements of some set Ω_A of observables for A.

A *provenance system* (P_A, Ω_A) extends a provenance framework by fixing a provenance interpretation and observations $P_A : \mathcal{T} \to \Omega_A$ for each principal A. We will focus on the case where there is just one principal, and sometimes omit subscripts. We often just write P and leave its range Ω implicit.

We define a quasiorder (reflexive and transitive, but not antisymmetric) on provenance views $P : \mathcal{T} \to \Omega, P' : \mathcal{T} \to \Omega'$ by defining $P \sqsubseteq P'$ to mean that there exists a function $h : \Omega' \to \Omega$ such that $h \circ P' = P$. Modulo isomorphism, this has a semilattice structure with least element $\bot : \mathcal{T} \to \{\star\}$ (for some constant $\star$) and greatest element $\top = \lambda t.t$. The least upper bound of $P_1 : \mathcal{T} \to \Omega_1$ and $P_2 : \mathcal{T} \to \Omega_2$ is given by $P_1 \sqcup P_2 : \mathcal{T} \to \Omega_1 \times \Omega_2$.

We now develop *provenance policies* that given systems may or may not satisfy. We write K_A or just K for the *initial knowledge* of principal A. This is a set of possible worlds, that is, traces, as understood by A. This could just be $\mathcal{T}$ itself, or could be a subset reflecting knowledge the user has already acquired about the possible behaviors of the system. The only requirement is that $\mathcal{T}_0 \subseteq K_A \subseteq \mathcal{T}$.

Now, we define the *knowledge of A after observation of ω* to be:

$$\hat{K}_A(\omega) = \{t' \in K_A \mid P_A(t') = \omega\}$$

We define a policy $\mathcal{P}$ to be a collection of pairs (K_A, Φ_A) where $\Phi_A \subseteq \mathcal{P}(\mathcal{T})$ is a set of sets of traces, for each A. Each element ϕ of Φ_A can be thought of as a set of worlds which the policy permits A to consider possible after making an observation. Formally, we say that a provenance system *satisfies* policy (K_A, Φ_A) if:

$$\forall A. \forall t \in \mathcal{T}_0. \hat{K}_A(P_A(t)) \in \Phi_A$$

Given a system, we define a policy that it enforces as follows. For each A, define K_A as $\mathcal{T}$ and define Φ_A as $\{\{t' \in \mathcal{T} \mid P(t) = P(t')\} \mid t \in \mathcal{T}_0\}$. We say that a provenance policy is *enforceable* if there is some provenance system that satisfies it, and *definable* if there is a provenance system that realizes it exactly.

Given a policy, we say that it *partitions* $\mathcal{T}_0$ if the set $\{S \cap \mathcal{T}_0 \mid S \in \Phi\}$ is a partition of $\mathcal{T}_0$. It is not difficult to show that:

Theorem 1. *A provenance policy is enforceable if and only if a subset of it partitions $\mathcal{T}_0$, and it is definable if and only if it partitions $\mathcal{T}_0$.*

A. Disclosure and obfuscation

We now propose formal definitions of two basic provenance security problems. In the following discussion, fix a provenance system.

Given principal A and query Q, we define the *provenance disclosure problem*, meaning the problem of determining whether $P_A(t) = P_A(t')$ implies $Q(t) = Q(t')$ for every $t, t' \in \mathcal{T}$. In words, this means that if two traces have the same provenance as viewed by A, then they have the same value with respect to Q. That is, the provenance query is (in principle at least) answerable from P_A.

Similarly, given query Q and principal A, we define the *provenance obfuscation problem* of determining that Q can never be answered using P_A. This means that for every trace $t \in \mathcal{T}$, there exists a trace t' such that $P_A(t) = P_A(t')$ yet $Q(t) \neq Q(t')$.

Lemma 1. *Disclosure and obfuscation are mutually exclusive: that is, a query cannot be both disclosed to A and obfuscated for A. However, they are not complementary.*

Proof: The first part is straightforward. For the second, consider $T = \{1, 2, 3\}$, $\Omega_A = \{even, odd\}$ and $P_A = \{(1, odd), (2, even), (3, odd)\}$. This system does not

[1] Note that this is simply a long-winded way of saying that $\mathcal{O}$ is a category. Likewise, $\mathcal{P}$ is a cartesian subcategory of the coslice category $\mathcal{T} \downarrow \mathcal{O}$. Category theory is not needed for understanding the technical results but could be used here to shorten notation.

disclose $Q = \{odd\}$ because both 1 and 3 map to odd, but it also does not obfuscate Q because when $t = 2$ we can always infer $t \notin Q$ from $P_A(t) = even$. ∎

The next few lemmas explore the disclosure property, which has several convenient properties stated in terms of the information ordering:

Lemma 2. *If $P \sqsubseteq P'$ and P discloses Q then P' discloses Q.*

Proof: Suppose P discloses Q and choose $h : \Omega' \to \Omega$ with $h \circ P' = P$. Let t, t' be given and assume $P'(t) = P'(t')$. Then $P(t) = h(P'(t)) = h(P'(t')) = P(t')$ so $Q(t) = Q(t')$ since P discloses Q. ∎

Corollary 1. *The top query $\top$ discloses any Q. If P and P' disclose Q then $P \sqcup P'$ discloses Q.*

Proof: The first parts are straightforward. The proof for $P \sqcup P'$ is immediate using the previous lemmas. ∎

Now we turn to properties of obfuscation. Obfuscation is antitone (that is, preserved by decreasing information), and the bottom element obviously obfuscates all queries.

Lemma 3. *If $P \sqsubseteq P'$ and P' obfuscates Q then so does P.*

Proof: Assume P' obfuscates Q and choose $h : \Omega' \to \Omega$ with $h \circ P' = P$. Let t be given. Since P' obfuscates Q there is a t' satisfying $P'(t) = P'(t')$ and $Q(t) \neq Q(t')$. Hence, $P(t) = h(P(t')) = h(P'(t')) = P(t')$ also holds, so P obfuscates Q. ∎

We consider the following decision problems involving disclosure and obfuscation, with respect to a given provenance framework.

1) *Disclosure checking:* Given a query Q and framework PSys, determine whether Q is disclosed to A by P_A. Then we say that $\mathsf{PSys} \models \mathsf{disclose}(Q, A)$.
2) *Obfuscation checking:* Given a query Q and framework PSys, determine whether Q is obfuscated by A. Then we say that $\mathsf{PSys} \models \mathsf{obfusc}(Q, A)$.

Each of these can also be considered as an algorithmic problem where the goal is to construct a provenance system that discloses or obfuscates a query. Also, the problem of checking simultaneous satisfiability of Boolean combinations of constraints could be considered, but as we will see already the above problems give us plenty to talk about.

B. Provenance-independent queries and policies

Later in the paper we focus on an important special case: understanding how provenance can convey information about the conventional system behavior. For example, often the behavior of the system is described by an input-output function or relation, assigned via a semantics interpreting programs to functions or relations. Principals are usually able to observe some or all inputs or outputs, and it may be more natural to frame policies involving provenance in terms of these concepts that usually already exist rather than in terms of a novel trace mechanism.

A provenance framework *has observable input and output* (or is an IO-framework) if there exist observables IN and OUT describing the input and output sets, together with provenance views $\mathsf{in} : \mathcal{T} \to \mathsf{IN}$ and $\mathsf{out} : \mathcal{T} \to \mathsf{OUT}$ that provide the input and output from a trace.

In an IO-framework, we can define queries that are *trace-invariant*. A trace-insensitive query is a property of traces that actually only depends on the input and output. In that case, we may abuse notation by considering such a query to be a subset $Q \subseteq \mathsf{IN} \times \mathsf{OUT}$, which can be implicitly identified with the set of traces $\{t \in \mathcal{T} \mid (\mathsf{in}(t), \mathsf{out}(t)) \in Q\}$. Similarly, we define a *trace-invariant view* $P_A : \mathcal{T} \to \Omega_A$ as one that is answerable from $\mathsf{in} \times \mathsf{out}$, and we define *trace-invariant policies* (K_A, Φ_A) where $K_A \subseteq \mathsf{IN} \times \mathsf{OUT}$ and $\Phi_A \subseteq \mathcal{P}(\mathsf{IN} \times \mathsf{OUT})$, which we identify with the corresponding trace-based policies in the obvious way.

We say that A *knows the input* (respectively, *output*) if $\mathsf{in} \sqsubseteq P_A$ (or respectively $\mathsf{out} \sqsubseteq P_A$).

We define the knowledge gain of A after observation ω given as follows:

$$\bar{K}_A(\omega) = \{(\mathsf{in}(t), \mathsf{out}(t)) \in K_A \mid P_A(t) = \omega\}$$

Moreover, a trace-invariant policy is satisfied in a provenance system if

$$\forall A. \forall t. \bar{K}_A(P(t)) \in \Phi_A$$

This condition is almost the same as for ordinary policies; only the types are different.

Note that a provenance system can provide useful information even if we only care about queries and policies that are trace-invariant. An example is given in Theorem 17.

Consider a setting where a principal knows the query and output, but does not know the input. Then the principal can learn about the input through observing provenance. Moreover, an *input query* is any query that is answerable using only $\mathsf{in}(t)$. We may want to ensure disclosure of certain input queries while requiring that others are obfuscated.

Consider a setting where a principal knows the input and output, but not the exact semantics of the system — that is, some of the traces considered possible by A are never actually displayed by the system. By observing provenance, the principal may be able to learn about the program.

Or, the principal may not be able to narrow down the set of possible programs, but may still be able to use the provenance to improve its knowledge of how the query would evaluate other inputs to other outputs.

Finally, given certainty about the program and input, the principal might still be uncertain that a given output is the *only* possible output (or if not, what are the possibilities). This can only happen if the underlying model of computation is nondeterministic. So we consider the knowledge about the output gained by providing provenance.

We will consider the following policy decision problems over trace-invariant frameworks:

1) *Input disclosure:* Assuming A knows the program and output, determine whether P_A discloses a given input query $Q \subseteq \mathsf{IN}$.
2) *Input obfuscation:* Assuming A knows the program and output, determine whether P_A obfuscates a given input query $Q \subseteq \mathsf{IN}$.
3) *Output disclosure:* Assuming A knows the program and output, determine whether P_A discloses a given output query $Q \subseteq \mathsf{OUT}$.
4) *Output obfuscation:* Assuming A knows the program and input, determine whether P_A obfuscates a given input query $Q \subseteq \mathsf{OUT}$.

Note that output disclosure and obfuscation problems are easy in the common case where programs are deterministic: A can simply rerun the program and check the answer. For this reason we will mostly focus on input disclosure and obfuscation in this paper, but the output problems are also of interest when the computation is nondeterministic.

III. Examples

The above framework is too abstract to offer much traction on specific computational situations. In order to study algorithms or mechanisms for analyzing or constructing provenance policies we need to provide additional detail about the components of provenance frameworks. We consider three instances:

- Sequential traces of finite automata
- Workflow graphs annotated with values
- Annotated relational queries

In each case, the sets of traces and provenance transformations can be described computationally using different formal languages, making it possible for us to study complexity and decidability issues in the next section. We rely on standard facts about regular languages and finite automata, but provide more details about the workflow graphs and database settings.

A. Sequential traces

We first consider a provenance framework **AUT** where the traces are sets of sequences $(Q \cup \Sigma)^*$. These, intuitively, describe the sequences of states and transitions of a given finite automaton M with alphabet Σ and states Q. For example, consider a two-state automaton that checks whether the length of a sequence is even. The realizable traces of any finite automaton again form a regular language, in this example it is $q_0((0+1)q_1(0+1)q_0)^*$.

Note that in this framework, we do not introduce any nonstandard concepts, since the standard notion of "run" of a finite automaton already seems adequate as a full explanation why the automaton accepts a string.

Sequential traces generated by automata lead to a provenance framework **AUT** where the queries are regular sets and the provenance views are given by the category of regular sets and transducer-definable functions. This framework also has input and output observations, obtainable by discarding the states from the trace (to obtain the input) and by checking the final state is an accept state (to obtain the output).

B. Workflow provenance graphs

We consider another form of traces based on graphical models of provenance as employed in a number of scientific workflow systems (e.g. Kepler [8] and Taverna [7] among many others); the Open Provenance Model [14] is a popular format for this kind of provenance but in this paper, we will focus on a simple form of OPM-like graphs. Our presentation draws primarily on the model introduced by Davidson et al. [**?**] which interprets workflow graphs (essentially) as straight-line code; this semantics for workflow graphs was also explored in more detail in [27] and related to structural causal models, but we will not pursue this connection in this paper.

Traces consist of directed acyclic graphs annotated with labels defining processes, input and output values. More formally, a graph is described by a set of processes P, disjoint sets of input and output ports I and O respectively, a function mapping each port to its associated process $p : I \cup O \to P$, and an edge relation $E \subseteq O \times I$ linking some output ports to input ports, such that each input is linked to at most one output. (We could also model E as a partial function from inputs to outputs.) We expect the graph to be acyclic in the obvious sense. A *trace* is a graph equipped with an additional function mapping each port to its value at run time, such that $(i, o) \in E$ implies $v(i) = v(o)$. The *external inputs* of the graph are the input ports that are not linked to any output and the *external outputs* are those output ports that are not linked to any input.

We consider simple queries of the form $Q(a, x) = \{v \mid v(a) = x\}$ that simply ask for the value of a given port. Moreover, we consider views of the form $v|_V$ where $V \subseteq I \cup O$, that is, v restricted to port names from V.

We can impose a category structure on these views where the objects are simply the sets V and the arrows are reverse inclusions. Thus, the most informative V is $I \cup O$, which leaves all values in place, and the least informative V is the empty set, which erases all information about the values encountered at run time, yielding the empty relation. This provenance framework has input and output as well, definable as projection to the external inputs and external outputs respectively.

Workflows may be *uninterpreted*, that is, arbitrary graphs labeled with arbitrary values, or they may adhere to some *interpretation* fixing the input-output relationships between the inputs and outputs of a given process. Formally, for each process p with inputs $I_p = \{i_1, \ldots, i_n\}$ and outputs $O_p = \{o_1, \ldots, o_m\}$, let $F_p : Val^{I_p} \to Val^{O_p}$ be a given

function; then we say that a trace v is consistent with this interpretation if for each p, we have $F_p(v(i_1), \ldots, v(i_n)) = (v(o_1), \ldots, v(o_m))$. Nondeterminism can easily be accommodated by weakening the functions F_p to relations R_p.

We write $\mathbf{WF}(G)$ be the provenance IO-framework generated by a workflow graph G. If in addition we expect the traces of G to obey a functional or relational interpretation we write $\mathbf{WF}(G, F)$ or $\mathbf{WF}(G, R)$ for the resulting framework.

C. Annotated relations

The semiring-annotated database model introduced by Green et al. [13] provides a convenient abstraction of a number of forms of provenance, including lineage [28], why-provenance [12], and how-provenance [13]. It can also be mapped to our framework. Let $\mathcal{U} = \{a, b, c, \ldots\}$ be a set of field names and finite sets $U, V \subseteq \mathcal{U}$ be called *sorts*. Let D be a set of data values. A record over U is a function from U to D, sometimes written as a set of pairs $(a_1 : d_1, \ldots)$. We write $t[U]$ for the record obtained by restricting t to field names from U, and $t[a/b]$ for the record obtained by renaming a in t to b.

Consider the syntax of positive relational algebra queries, given as follows:

$$q \quad ::= \quad R \mid \sigma_{a=b}(q) \mid \pi_V(q) \mid \rho_{a/b}(q) \mid q \cup q' \mid q \bowtie q'$$

Here R is a relation variable name, $\sigma_{a=b}(q)$ selects records whose a and b fields are equal, $\pi_V(q)$ projects the records in q down to the fields mentioned in V, and $\rho_{a/b}(q)$ renames records. Also, $\cup$ and $\bowtie$ are the relational union and join operations respectively. A join simply considers all pairs of records from the two input relations (which may have different sorts) and merges all of the compatible records. If the sorts of the arguments are disjoint then join is the same as the cartesian product of the two relations. Query expressions can be sort-checked in a straightforward way; for example, if $R : U$ and $S : V$ then $\sigma_{a=b}(R) : U$ if $a, b \in U$, and $\pi_V(R) : V$ if $V \subseteq U$, and $R \cup S : U$ if $U = V$, and $R \bowtie S : U \cup V$.

Database queries are normally interpreted over finite relations, that is, finite sets of records having the same type U. Any such relation can be thought of as a function from records over U to $\mathbb{B}$, that is, as a characteristic function. For finite relations, characteristic functions have a *finite support* property, that is, $r(x) = 0$ for all but finitely many records x. Moreover, finite multisets can be viewed as finitely-supported functions from records to $\mathbb{N}$. Green et al. [13] observed that this approach can be generalized further so that for any commutative semiring K, we can view a function from tuples to K as a table in which each row is annotated with a nonzero element of K. (An annotation with value 0 represents a tuple not present in the table.)

Recall that a commutative semiring is a structure $(K, 0, 1, +, \cdot)$ such that $(K, 0, +)$ and $(K, 1, \cdot)$ are commutative monoids, that $\cdot$ distributes over $+$ and that 0 annihilates $\cdot$. From now on we will just write *semiring* instead of commutative semiring. Typical examples of semirings include the natural numbers $\mathbb{N}$, the Booleans $\mathbb{B}$, and the free semiring $\mathbb{N}[X]$ consisting of polynomials with coefficients from $\mathbb{N}$ and generators X.

Queries can be evaluated over *annotated databases* where the annotations are drawn from a commutative semiring. A positive database query can be interpreted as a function taking relation-valued variables and yielding a relation. For each finite set of attribute names U and semiring K, we model U-ary relations as finitely-supported functions:

$$\begin{aligned} Rel(U) &= D^U \to_{fs} K \\ &= \{r : D^U \to \mathbb{B} \mid \{t \mid r(t) \neq 0\} \text{ is finite}\} \end{aligned}$$

Specifically, the positive relational operations over K-annotated relations are interpreted as follows:

$$\begin{aligned} \sigma^K_{a=b}(r) &= \lambda t.\mathsf{if}\ t.a = t.b\ \mathsf{then}\ r(t)\ \mathsf{else}\ 0 \\ \pi^K_V(r) &= \lambda t. \sum_{u:U, u[V]=t} r(u) \\ \rho^K_{a/b}(r) &= \lambda t.r(t(b/a)) \\ r \cup^K s &= \lambda t.r(t) + s(t) \\ r \bowtie^K s &= \lambda t.r(t[U]) \cdot s(t[V]) \end{aligned}$$

Here, we assume that $r : U$ and $s : V$ and we assume that the relational expressions are well-sorted as discussed above. In the equation for projection, note that the sum is well-defined since there are at most finitely many u in the domain of r such that $r(u) \neq 0$.

Let q be a fixed, positive query. The set of *traces* is the set of pairs $(DB, q^{\mathbb{N}[X]}(DB))$ where DB is a distinctly annotated database instance with annotations from X, and $q^{\mathbb{N}[X]}(DB)$ is the result of evaluating query expression q on DB using the $\mathbb{N}[X]$-valued relational operations defined above.

The *queries* we are concerned with in this setting will simply be Boolean relational queries over the input database and result. The reason we limit attention here to queries that can be answered only from the ordinary relations is that very little is known so far about Boolean queries over K-relations.

The *observables* will consist of collections K-$\mathbf{REL}$ of semiring-valued input-output pairs for each K and mappings among them generated by homomorphisms among semirings. Then the category of provenance views $P : \mathcal{T} \to K$-$\mathbf{REL}$ will essentially be generated by homomorphisms from $\mathbb{N}[X] \to K$, which in turn are uniquely defined by functions from $X \to K$.

The conventional input and output of a query can be observed by first applying the semiring homomorphism from $\mathbb{N}[X]$ to $\mathbb{B}$ and then projecting the input component or respectively the output component. This yields an IO-framework.

IV. Main Results

A. Sequential traces

We first consider the disclosure and obfuscation problems for sequential traces. Here, some positive results can be obtained using standard elements of the theory of finite automata and transducers.

Theorem 2. *The disclosure checking problem for* $\mathbf{AUT}$ *is decidable.*

Proof: Given Q and P, realized by an automaton and transducer respectively, we form transducers T and T' such that T behaves as P restricted to Q while T' behaves as P restricted to $\mathcal{T} - Q$. Then the range of T is the regular set X of possible P-values of elements of Q and that of T' is the regular set X' of possible P-values of traces not in Q. Then P discloses Q if and only if $X \cap X'$ is empty, which is decidable. ∎

Corollary 2. *Input and output disclosure are decidable for* $\mathbf{AUT}$.

Theorem 3. *The obfuscation checking problem is decidable if P has a finite range.*

Proof: Given Q and $P : \mathcal{T} \to \Omega$, if Ω is finite then we can enumerate all pre-images $P^{-1}[\omega]$ of observations ω. These are regular, so for each ω we can decide whether $P^{-1}[\omega] \subseteq Q$ or $P^{-1}[\omega] \subseteq \mathcal{T} - Q$. If this is the case for any t, then we know that P fails to obfuscate Q, otherwise P does obfuscate Q. ∎

Corollary 3. *Output obfuscation is decidable for* $\mathbf{AUT}$.

It is not clear whether obfuscation checking (even restricted to input queries) is decidable in general for $\mathbf{AUT}$ so we leave this as an open question.

B. Provenance graphs

We now turn to the provenance framework $\mathbf{WF}(G)$ of runs of a given workflow graph G. In the case where the workflow semantics is unknown, both disclosure and obfuscation turn out to be very easy:

Theorem 4. *Disclosure and obfuscation are decidable in polynomial time for an unrestricted workflow graph.*

Proof: For unrestricted interpretations, we first assume that each a can assume two or more values. A query $Q(a, x)$ is disclosed by V precisely when $a \in V$ or some port linked by an edge to a in E is in V. Since there are no restrictions on traces v other than compatibility with E, as soon as a is hidden we cannot tell whether $v(a) = x$. Similarly, a query $Q(a, x)$ is obfuscated by V precisely when a and all of the ports linked to it by E are hidden in V, because as soon these ports are hidden, we can be sure that there is another trace that is equivalent modulo V but not modulo Q. If a given a has only one possible value then it is always disclosed and cannot be obfuscated. ∎

This suggests that these problems are only really interesting in the (more realistic) situation where the principal has some domain knowledge about the workflow's semantics. To investigate the complexity of disclosure and obfuscation in the presence of an interpretation, we need to restrict attention to some computationally sensible class of interpretations. We will consider workflows that are essentially Boolean circuits. This means that each process is interpreted as a gate such as conjunction, disjunction or negation, and has exactly one output. In general, workflows could be much more complex than Boolean circuits, but studying this special case will give us lower bounds for the general case. We write $\mathbf{WF}(G, \mathbb{B})$ for a workflow setting where the processing steps are interpreted as Boolean gates.

Theorem 5. *Disclosure and obfuscation checking are decidable for Boolean circuit graph settings* $\mathbf{WF}(G, \mathbb{B})$.

Proof: There are finitely many (but doubly exponential) sets of possible valuations. So, we can decide whether a given Q is disclosed by enumerating all possible valuations consistent with each observation induced by the given view V, and checking that all of them either satisfy Q or all of them satisfy its complement. The reasoning for obfuscation checking is similar. ∎

Theorem 6. *Output disclosure checking is trivial for any* $\mathbf{WF}(G, F)$.

Proof: Given an output query, if the input is known then we can evaluate the output query by evaluating the circuit on the input, and then checking whether the output query is satisfied. That is, any query is always disclosed once we know the input. ∎

Theorem 7. *Input disclosure checking is NP-hard and $coNP$-hard for Boolean circuit workflow settings* $\mathbf{WF}(G, \mathbb{B})$.

Proof: Given a 3CNF instance C, add nodes a and b where $b = C \vee a$. Then the output view is just b and the input query we are interested in is $Q = (a = 1)$. Now, if C is unsatisfiable, we can clearly determine whether $a = 1$ holds from observing b, since then $b = 0 \vee a = a$. Conversely, if we can always determine whether $a = 1$ holds by observing b, then C cannot be satisfiable, since if C has a satisfying assignment then there are two valuations v, v' matching C but with $v(a) = 0$ and $v'(a) = 1$ and $v(b) = 1 = v(0)$. A similar argument suffices for $coNP$-hardness, using conjunction instead of disjunction. ∎

Corollary 4. *Disclosure checking is also NP-hard and $coNP$-hard.*

Now we turn to obfuscation checking. As before, output obfuscation checking is trivial since we can always just rerun

the graph and check whether the output satisfies the query.

Theorem 8. *The output obfuscation problem is trivial for Boolean circuit settings* $\mathbf{WF}(G, F)$.

Proof: Similar to the proof for output disclosure checking, due to determinacy. An output query can never be obfuscated if the input and program are known. ■

Theorem 9. *Input obfuscation is* NP*-hard and* $coNP$*-hard for Boolean circuit setting* $\mathbf{WF}(G, F)$.

Proof: The proof is similar to that for disclosure checking. For NP-hardness, given a circuit C we build $b = a \wedge C$ and also consider all of the outputs of C to be observable outputs by copying. Then if C is unsatisfiable then a is obfuscated since b will always be 0 and a can be anything. Conversely, if a is obfuscated then C can never evaluate to 1 since in that case we would be able to infer that $a = 1$ from b. Note that it is important to treat the inputs to C as part of the output in order to ensure that $a = 1$ is inferrable from $b = 1$. This is not an imposition since we can for example simply add output nodes connected to the inputs to C by two negation gates. ■

Finally, we note that if we consider a relational interpretation for processes instead of a functional one, we can get the same complexity lower bounds for both output and input obfuscation checking. The reason is that if we can use relations, then the problems are completely symmetric in input and output: we can reuse the same ideas in the proofs above, just reversing the directions of the circuits. This shows that nondeterministic output disclosure or obfuscation checking is nontrivial in general.

C. Annotated relations

Now we turn to the problem of deciding disclosure and obfuscation problems over annotated relations. In this section we will consider the problem of determining whether a given Q is disclosed or obfuscated by specific classes of provenance views. The reason for doing this is that the space of possible provenance views (i.e., semirings and homomorphisms) is large, and it is not clear how they should be represented in general. We also assume that the underlying relational query that is used to generate the outputs from the inputs is known to principals.

As before, output disclosure is trivial since the queries are deterministic, so we say no more about it. Input disclosure and obfuscation turn out to be rather difficult to analyze for relational queries, especially if infinite data domains are considered. If the domain is finite, then these properties are at least decidable:

Theorem 10. *Input disclosure and obfuscation are decidable for relational queries over a finite domain, for any computable semiring provenance interpretation.*

Proof: There are at most finitely many input relations possible over a finite domain, so we can evaluate all of them and group them by equal output, and then check whether a given input query is disclosed or obfuscated by evaluating it on all of the grouped sets. ■

We will now restrict attention to a simple case that nevertheless helps illustrate the main ideas. We consider input queries that ask for the presence or absence of a single tuple, often just written as $R(t)$. We also consider the disclosure or obfuscation behavior of queries that involve a single relational step, using inputs that are distinct variables.

First, we consider what is inferrable about the input without any assistance from provenance: namely, the disclosure and obfuscation properties of the relational operators in the classical $\mathbb{B}$-valued semantics.

Theorem 11. *Input tuple disclosure with respect to* $\mathbb{B}$ *is characterized as follows for individual operators:*

1) R *discloses* $R(t)$ *for any* $R(t)$.
2) $\sigma_{a=b}(R)$ *discloses* $R(t)$ *if and only if* $t.a = t.b$.
3) $\pi_V(R)$ *discloses* $R(t)$ *if and only if* $R : V$.
4) $\rho_{a/b}(R)$ *always discloses* $R(t)$.
5) $R \cup S$ *does not disclose* $R(t)$ *for any* t.
6) $R \bowtie S$ *does not disclose* $R(t)$ *for any* t.

Proof: For each disclosure case, it suffices to exhibit a way to find tuple t' that is in the output if and only if $R(t)$ is in the input for some t'. In cases (1) and (2), $t' = t$, and in case (2) we need the assumption that $t.a = t.b$ because otherwise t will not appear in the output. In case (4), $t' = t(a/b)$. For case (3), if V includes all the attributes of R then the projection is essentially the same as just returning R. Conversely, if $V \subset U$ where $R : U$ then (assuming data domain with at least two elements) we can always find two instantiations for R that project to the same value but such that one contains t and the other does not. For union, we cannot tell whether an occurrence of t in the output comes from R, S or both. In the final case for join, we cannot tell whether the absence of a tuple t' with $t'[U] = t$ in the output is due to the absence of t from R or to the absence of a matching tuple $t'[V]$ in S. ■

Theorem 12. *Input obfuscation with respect to* $\mathbb{B}$ *is characterized as follows for individual relational operators:*

1) R *does not obfuscate* $R(t)$ *for any* $R(t)$.
2) $\sigma_{a=b}(R)$ *obfuscates* $R(t)$ *if and only if* $t.a \neq t.b$.
3) $\pi_V(R)$ *does not obfuscate* $R(t)$.
4) $\rho_{a/b}(R)$ *does not obfuscate* $R(t)$ *for any* $R(t)$.
5) $R \cup S$ *does not obfuscate* $R(t)$ *for any* $R(t)$.
6) $R \bowtie S$ *does not obfuscate* $R(t)$ *for any* $R(t)$.

Proof: The first and fourth cases are obvious since the operations are invertible. In the second, assume $t.a \neq t.b$. Then $\sigma_{a=b}(R)$ cannot distinguish between $R = \emptyset$ and $R = \{t\}$. In the third case, if $R = \emptyset$ then whenever $\pi_V(R) = \emptyset$ we have $t \notin R$. In the fifth case, if $R = \emptyset = S$ then we can determine that $t \notin R$ from the fact that $R \cup S = \emptyset$.

Similarly, in the sixth case we can deduce that $t \in R$ from the presence of any tuples t' with $t'[U] = t$ in $R \bowtie S$. ■

Now we consider disclosure and obfuscation with respect to the view obtained by taking the $\mathbb{N}[X]$-annotated output and discarding the input. So, the question is, what can we learn about the input solely by inspecting the $\mathbb{N}[X]$-annotations on the output, and without necessarily knowing which annotations from X correspond to which tuples in the input? Perhaps surprisingly, the presence of provenance annotations on their own does not seem to increase what we can infer about the input, at least if we consider one-step queries:

Theorem 13. *Input tuple disclosure with respect to $\mathbb{N}[X]$ is characterized as follows for individual operators:*

1) *R discloses $R(t)$ for any $R(t)$.*
2) *$\sigma_{a=b}(R)$ discloses $R(t)$ if and only if $t.a = t.b$.*
3) *$\pi_V(R)$ discloses $R(t)$ if and only if $R : V$.*
4) *$\rho_{a/b}(R)$ always discloses $R(t)$.*
5) *$R \cup S$ does not disclose $R(t)$ for any t.*
6) *$R \bowtie S$ does not disclose $R(t)$ for any t.*

Proof: The proofs of (1), (2), (3), and (4) are the same as before. For part (5), if the output contains t with an atomic annotation x, we cannot determine whether t is present in R or S. Similarly, for part (6), if the output does not contain a tuple extending t, we cannot tell whether this is because t is not in R or because there is no matching tuple in S, because the result does not contain any annotations generated by t. ■

Theorem 14. *Input obfuscation with respect to $\mathbb{N}[X]$ is characterized as follows for individual relational operators:*

1) *R does not obfuscate $R(t)$ for any $R(t)$.*
2) *$\sigma_{a=b}(R)$ obfuscates $R(t)$ if and only if $t.a \neq t.b$.*
3) *$\pi_V(R)$ does not obfuscate $R(t)$.*
4) *$\rho_{a/b}(R)$ does not obfuscate $R(t)$ for any $R(t)$.*
5) *$R \cup S$ does not obfuscate $R(t)$ for any $R(t)$.*
6) *$R \bowtie S$ does not obfuscate $R(t)$ for any $R(t)$.*

Proof: Most cases are straightforward or similar to previous cases. ■

Finally, we consider disclosure problems involving provenance views mapping $\mathbb{N}[X]$ to $\mathbb{B}$, where we first apply the homomorphism (possibly employing information about the input annotations) and then discard the input. Such a homomorphism is defined uniquely by a function $X \to \mathbb{B}$, that is, a subset of X. Moreover, since each X corresponds to a unique tuple in the input, we can represent such a homomorphism by a collection of Boolean queries over the input tables. Thus, if the input tables are $r_1, \ldots, r_n$ then we say that $h : \mathbb{N}[X] \to \mathbb{B}$ is *represented* by Boolean queries $q_1, \ldots, q_n$ provided that $h(r_i(t)) = q_i(t)$.

Another way of thinking about this setting is to imagine we have a database with some input tuples marked secret (mapped to 0 by h) and others marked public (mapped to 1 by h). Then the provenance view is the result produced by the query when restricted to public input tuples. This may be different from (but must be contained in) the real output. This approach allows some limited use to be made of the connection between the annotations in the input and the result: although A cannot directly view the input, her view is allowed to set the annotations to Boolean values based on the result of an input query and tell A the result that the underlying database query would have if run on the public inputs.

Theorem 15. *The input disclosure problem for homomorphisms to $\mathbb{B}$ is characterized as follows for individual operators:*

1) *R discloses $R(t)$ for any $R(t)$.*
2) *$\sigma_{a=b}(R)$ discloses $R(t)$ if and only if $t.a = t.b$.*
3) *$\pi_V(R)$ discloses $R(t)$ if and only if for any u. $h(r(u)) = 1$ and $u[V] = t[V]$ implies $u = t$.*
4) *$\rho_{a/b}(R)$ always discloses $R(t)$.*
5) *$R \cup S$ discloses $R(t)$ if and only if $h(r(t)) = 1$ and $h(s(t)) = 0$.*
6) *$R \bowtie S$ does not disclose $R(t)$ for any $R(t)$.*

Proof: Parts (1,2,4) are similar to previous arguments.

For part (3), we need to show that if $R(t)$ is disclosed by $\pi_V(R)$ then $R(t)$ is the only tuple selected by the homomorphism that can project to $t[V]$. If there is more than one such tuple then we could violate disclosure using it. If h does not accept any tuples that are equivalent to t modulo projection to V then we cannot be certain which of these tuples was actually present in the input. Thus, if the projection discloses $R(t)$ then h must select $R(t)$ and no other tuple $R(u)$ with $t[V] = u[V]$. The reverse direction is straightforward.

For part (5), suppose that $R \cup S$ discloses $R(t)$. Then we know that $h(r(t)) \vee h(s(t)) = h(r'(t)) \vee h(s'(t))$ implies $r(t) = r'(t)$, for any r, s, r', s' providing values for R, S. Restricting attention to the Boolean values of t in r, s, r', s', we can see there are sixteen possibilities. If $h(s(t))$ is nonzero then we can obtain a counterexample to the disclosure assumption, and similarly if $h(r(t))$ is zero. Conversely, clearly taking $h(r(t)) = 1$ and $h(s(t)) = 0$ suffices to ensure that h discloses $R(t)$ since it ensures that the provenance value tells us the contents of R only.

For part (6), we need to confirm that a clever choice of homomorphism is still not enough to be able to determine whether $R(t)$ is in the input by looking at $R \bowtie S$. Let h be given and choose $t : U \cup V$ in the range of $R \bowtie S$. Consider $r = \{t[U]\}, s = \emptyset, r' = \emptyset, s' = \{t[V]\}$. Then the joins $r \bowtie s$ and $r' \bowtie s'$ have the same result $\emptyset$ and their h-values are also equal, but $r(t)$ holds and $r'(t)$ does not. ■

Theorem 16. *The input obfuscation problem for homomorphisms to $\mathbb{B}$ is is characterized as follows for individual relational operators:*

1) R *does not obfuscate* $R(t)$ *for any* $R(t)$.
2) $\sigma_{a=b}(R)$ *obfuscates* $R(t)$ *if and only if* $t.a \neq t.b$.
3) $\pi_V(R)$ *does not obfuscate* $R(t)$.
4) $\rho_{a/b}(R)$ *does not obfuscate* $R(t)$ *for any* $R(t)$.
5) $R \cup S$ *does not obfuscate* $R(t)$ *for any* $R(t)$.
6) $R \bowtie S$ *does not obfuscate* $R(t)$ *for any* $R(t)$.

Proof: Similar to previous arguments. Unlike for disclosure, we cannot make use of the homomorphism to improve obfuscation because we only have the ability to consider hypothetical deletions from the database, and such deletions do not tell us anything useful about obfuscation. ■

We conclude with a brief discussion of the implications of these results. As discussed above, it may seem surprising that simply providing the output with the most general possible semiring annotations does not provide more information about the input than the conventional $\mathbb{B}$-valued semantics (at least to a first approximation using single-step queries). This may be partly an artifact of the fact that the semiring model was not designed with input query answerability in mind. Nevertheless, it would be interesting to investigate the disclosure and obfuscation properties of other provenance models for databases, as well as to explore other forms of annotation propagation based on semirings or richer classes of input queries. These problems appear challenging. On the other hand, the results above show how provenance can be useful for enforcing policies that are stated only in terms of input-output behavior:

Theorem 17. *Consider the semiring provenance framework for query* $R \cup S$. *The input query* $R(t) \in DB$ *is disclosed by the* $\mathbb{B}$*-homomorphism generated by a query that accepts all tuples in* R *and rejects all tuples in* S.

Proof: This is essentially part (5) of Theorem 15. ■

V. Related and future work

There exists a large literature on provenance, and we direct readers to recent surveys and summaries such as Moreau [5] and the W3C Provenance Incubator Group report [4] for an introduction to the area, and to our previous paper [6] for an overview of open semantics and security problems involving provenance.

Work on database provenance is distinctive in that several different formal models have now been defined for database query languages with well-understood semantics. This makes it easier to compare, relate and generalize these approaches, though such comparisons are only starting to appear [13], [29]. For most of these models, there are semantic guarantees (or even exact semantic characterizations) relating the provenance records to the denotation of the program. However, even for the semiring model, basic questions such as query equivalence for annotated relations [30] and how to implement provenance and query provenance-annotated databases [31], [32] are only beginning to be addressed.

For workflow provenance, many implementations exist [7], [8] but formal definitions of the meaning of workflow programs have only started to appear recently (see for example [33], [34]), while the provenance semantics of these tools is usually specified informally, at best [7]. As a result there is a variety of models and styles of provenance for workflows. Recently, a community effort to develop a common data model for provenance has converged on the Open Provenance Model (OPM) [14], but so far this effort has focused on the syntax or structure of provenance graphs and not on their semantics or their relationship to the behavior of the systems that generate them. Our previous paper [27] made one proposal for interpreting OPM-style provenance graphs in terms of structural causal models [35], and Davidson et al. [25] adopt a similar model; however, both approaches stop well short of addressing the all of the features employed by workflow provenance systems.

Provenance security has been considered by a number of authors. Hasan et al. [16] and Braun et al. [17] set out a number of research questions, as does a more recent position paper by Davidson et al. [20]. Hasan et al. [10] develop a mechanism for securing the provenance describing a chain of revisions of a document (say, a file in a file system), using appropriate encryption and digital signatures to allow auditors to check the integrity of the provenance without necessarily having access to the underlying data or vice versa. Zhang et al. [18] develop tamper-detection techniques for provenance in databases. Cirillo et al. [36] study provenance policies based on logics of knowledge for distributed object calculi. Corcoran et al. [19] support provenance-tracking as one of a number of label-based security policies in a cross-tier Web programming language SELinks. However, most of this work focuses on applying existing security mechanisms to provenance data, rather than on understanding security policies involving provenance.

As far as we know, there is no prior proposal for a formal framework for provenance security that is as general as ours. Chong [22] proposed semantic definitions of provenance security policies, formulated in terms of a syntactic model of traces based on previous (unpublished) work by Cheney et al. [23]. Chong discussed simple policies expressing that either data or its provenance are high- and low-security, and formalized properties such as "provenance security" stating that the provenance of a run of a system is not inferrable from the data, versus "data security" stating that the high-security input data of a system is not inferrable from its provenance. Part of the motivation for the present paper is to generalize and abstract Chong's definitions beyond the small language originally considered. More recently, Lyle and Martin [37] gave a detailed comparative survey of topics in provenance and in security, pointing out many parallel developments. Some other topics in security, such as non-repudiation [38], plausible deniability or differential privacy [39], also appear analogous to our disclosure and ob-

fuscation properties, and this connection could be explored.

Some previous work on provenance (including work on where-provenance [40], how-provenance [13], [41], and dependency-provenance [24]) explicitly cites security motivations. Some other work, particularly that of Davidson et al. [25], proposes formal definitions of specific privacy (confidentiality) policies for workflow provenance but does not seek to provide a framework that applies to other provenance models. Conversely, provenance-like techniques seem to arise naturally in dynamic information-flow settings where labels are propagated dynamically in order to support more flexible enforcement of a security policy [19], [36], [42]. However, previous work on provenance has not been used directly to enforce ordinary security policies, nor have dynamic labeling techniques been employed as a way of implementing provenance (although dependency provenance comes close to this).

Our framework is defined primarily with *static* provenance situations in mind: that is, we consider a fixed, stateless system and consider what a principal can learn from observing some approximation of one run of the system. In particular, our model does not have much to say about *dynamic provenance* in extant systems that seek to record all changes to some shared data [26], or to maintain the integrity of provenance records in a stateful system (e.g. file system) and prevent forgery [10]. We believe that modeling this kind of situation will require further extensions to our model to permit principals to both read and write data and communicate with one another, and to model what can be learned by multiple interactions with the system or other principals.

Moreover, we assume that the system designer and other principals are not actively trying to mislead one another, or to corrupt or leak data. Thus, we leave out issues such as authentication, encryption, digital signatures/hashing and so on. While these assumptions are reasonable for this initial assay since it helps focus attention on the novel issues, it will obviously be important to relax them and develop an understanding of how provenance fits into the lager picture of system security.

VI. Conclusions

The question of how to define and implement effective provenance-tracking mechanisms has received a great deal of attention in the last few years. It appears likely that some new standards or infrastructures for managing provenance on the Web will be deployed as a result — indeed, the W3C has now commissioned a Provenance Working Group to standardize an interchange format for provenance. However, as several authors have pointed out already, this obviously introduces a number of concerns for security. In this paper, we have developed a relatively high-level and generic framework for provenance that abstracts away many of the distracting details of particular systems, and makes it possible to identify some commonalities and general properties of such systems. We have also explored three instances of this framework, based on regular expressions and transducers, simple workflows, and provenance-annotated relational queries. This exploration helps validate the model and justify its design rather than to provide new insight into existing approaches to provenance. Nevertheless, we believe that this first step represents progress towards a proper formalization of secure provenance.

Acknowledgment

The author is supported by a Royal Society University Research Fellowship.

References

[1] S. Carey and G. Rogow, "UAL shares fall as old story surfaces online," Wall Street Journal, September 2008, http://online.wsj.com/article/SB122088673738010213.html.

[2] G. Miller, "A scientist's nightmare: Software problem leads to five retractions," *Science*, vol. 314, no. 5807, pp. 1856–1857, December 2006.

[3] S. Varghese, "UK government gets bitten by Microsoft Word," Sydney Morning Herald, July 2003, http://www.smh.com.au/articles/2003/07/02/1056825430340.html.

[4] Y. Gil, J. Cheney, P. Groth, O. Hartig, S. Miles, L. Moreau, and P. Pinheiro da Silva et al., "Provenance XG final report," 2010.

[5] L. Moreau, "The foundations for provenance on the web," *Foundations and Trends in Web Science*, vol. 2, no. 2–3, 2010.

[6] J. Cheney, S. Chong, N. Foster, M. Seltzer, and S. Vansummeren, "Provenance: A future history," in *OOPSLA Companion (Onward! 2009)*, 2009, pp. 957–964.

[7] P. Missier, K. Belhajjame, J. Zhao, M. Roos, and C. A. Goble, "Data lineage model for Taverna workflows with lightweight annotation requirements," in *IPAW*, 2008, pp. 17–30.

[8] B. Ludäscher, I. Altintas, C. Berkley, D. Higgins, E. Jaeger, M. Jones, E. A. Lee, J. Tao, and Y. Zhao, "Scientific workflow management and the Kepler system." *Concurrency and Computation: Practice and Experience*, vol. 18, no. 10, pp. 1039–1065, 2006.

[9] O. Benjelloun, A. D. Sarma, A. Y. Halevy, M. Theobald, and J. Widom, "Databases with uncertainty and lineage," *VLDB J.*, vol. 17, no. 2, pp. 243–264, 2008.

[10] R. Hasan, R. Sion, and M. Winslett, "Preventing history forgery with secure provenance," *Trans. Storage*, vol. 5, pp. 12:1–12:43, December 2009.

[11] K.-K. Muniswamy-Reddy, D. A. Holland, U. Braun, and M. Seltzer, "Provenance-aware storage systems," in *USENIX Annual Technical Conference*. USENIX, June 2006, pp. 43–56.

[12] P. Buneman, S. Khanna, and W. Tan, "Why and where: A characterization of data provenance," in *ICDT*, ser. LNCS, no. 1973. Springer, 2001, pp. 316–330.

[13] T. J. Green, G. Karvounarakis, and V. Tannen, "Provenance semirings," in *PODS*. ACM, 2007, pp. 31–40.

[14] L. Moreau, B. Clifford, J. Freire, J. Futrelle, Y. Gil, P. Groth, N. Kwasnikowska, S. Miles, P. Missier, J. Myers, B. Plale, Y. Simmhan, E. Stephan, and J. Van den Bussche, "The open provenance model core specification (v1.1)," *Future Generation Computer Systems*, vol. 27, no. 6, pp. 743–756, 2011.

[15] O. Biton, S. C. Boulakia, S. B. Davidson, and C. S. Hara, "Querying and managing provenance through user views in scientific workflows," in *ICDE*. IEEE, 2008, pp. 1072–1081.

[16] R. Hasan, R. Sion, and M. Winslett, "Introducing secure provenance: problems and challenges," in *Proceedings of the 2007 ACM workshop on Storage security and survivability (StorageSS 2007)*. New York, NY, USA: ACM, 2007, pp. 13–18.

[17] U. Braun, A. Shinnar, and M. Seltzer, "Securing provenance," in *Proceedings of the 3rd conference on Hot topics in security*. Berkeley, CA, USA: USENIX Association, 2008, pp. 4:1–4:5.

[18] J. Zhang, A. Chapman, and K. Lefevre, "Do you know where your data's been? — tamper-evident database provenance," in *Proceedings of the 6th VLDB Workshop on Secure Data Management (SDM 2010)*. Berlin, Heidelberg: Springer-Verlag, 2009, pp. 17–32.

[19] B. J. Corcoran, N. Swamy, and M. Hicks, "Cross-tier, label-based security enforcement for web applications," in *SIGMOD*, 2009.

[20] S. B. Davidson, S. Khanna, S. Roy, and S. C. Boulakia, "Privacy issues in scientific workflow provenance," in *Proceedings of the 1st International Workshop on Workflow Approaches to New Data-centric Science (WANDS 2010)*. New York, NY, USA: ACM, 2010, pp. 3:1–3:6.

[21] P. Buneman, J. Cheney, W.-C. Tan, and S. Vansummeren, "Curated databases," in *Proceedings of the 2008 Symposium on Principles of Database Systems (PODS 2008)*, 2008, pp. 1–12.

[22] S. Chong, "Towards semantics for provenance security," in *Workshop on the Theory and Practice of Provenance*, J. Cheney, Ed. USENIX, 2009.

[23] J. Cheney, U. Acar, and A. Ahmed, "Provenance traces (extended report)," Tech. Rep., 2008, `arXiv:0812.0564v1`.

[24] J. Cheney, A. Ahmed, and U. A. Acar, "Provenance as dependency analysis," *Mathematical Structures in Computer Science*, 2011, in press.

[25] S. B. Davidson, S. Khanna, S. Roy, J. Stoyanovich, V. Tannen, and Y. Chen, "On provenance and privacy," in *Proceedings of the 14th International Conference on Database Theory (ICDT 2011)*. New York, NY, USA: ACM, 2011, pp. 3–10.

[26] P. Buneman, A. Chapman, and J. Cheney, "Provenance management in curated databases." in *SIGMOD*, 2006, pp. 539–550.

[27] J. Cheney, "Causality and the semantics of provenance," in *Proceedings of the 2010 Workshop on Developments in Computational Models*, 2010.

[28] Y. Cui, J. Widom, and J. L. Wiener, "Tracing the lineage of view data in a warehousing environment." *ACM Trans. Database Syst.*, vol. 25, no. 2, pp. 179–227, 2000.

[29] J. Cheney, L. Chiticariu, and W. C. Tan, "Provenance in databases: Why, how, and where," *Foundations and Trends in Databases*, vol. 1, no. 4, pp. 379–474, 2009.

[30] T. J. Green, "Containment of conjunctive queries on annotated relations," in *ICDT*, Saint Petersburg, Russia, March 2009.

[31] G. Karvounarakis, Z. G. Ives, and V. Tannen, "Querying data provenance," in *SIGMOD Conference*, 2010.

[32] B. Glavic and G. Alonso, "Perm: Processing provenance and data on the same data model through query rewriting," in *ICDE*. IEEE, 2009.

[33] J. Sroka, J. Hidders, P. Missier, and C. Goble, "A formal semantics for the Taverna 2 workflow model," *Journal of Computer and System Sciences*, vol. In Press, Corrected Proof, 2009.

[34] J. Hidders, N. Kwasnikowska, J. Sroka, J. Tyszkiewicz, and J. Van den Bussche, "A formal model of dataflow repositories," in *DILS*, ser. LNCS, vol. 4544. Springer, 2007, pp. 105–121.

[35] J. Pearl, *Causality*. Cambridge University Press, 2000.

[36] A. Cirillo, R. Jagadeesan, C. Pitcher, and J. Riely, "Tapido: Trust and authorization via provenance and integrity in distributed objects," in *ESOP*, 2008, pp. 208–223.

[37] J. Lyle and A. Martin, "Trusted computing and provenance: better together," in *Proceedings of the 2nd conference on Theory and practice of provenance (TAPP 2010)*. Berkeley, CA, USA: USENIX Association, 2010.

[38] S. Schneider, "Formal analysis of a non-repudiation protocol," in *Proceedings of the 11th IEEE workshop on Computer Security Foundations*. Washington, DC, USA: IEEE Computer Society, 1998, pp. 54–65.

[39] C. Dwork, "A firm foundation for private data analysis," *Commun. ACM*, vol. 54, pp. 86–95, January 2011.

[40] P. Buneman, J. Cheney, and S. Vansummeren, "On the expressiveness of implicit provenance in query and update languages," *ACM Transactions on Database Systems*, vol. 33, no. 4, p. A28, November 2008.

[41] J. N. Foster, T. J. Green, and V. Tannen, "Annotated XML: queries and provenance," in *PODS*, 2008, pp. 271–280.

[42] P. Shroff, S. F. Smith, and M. Thober, "Dynamic dependency monitoring to secure information flow," in *CSF*. IEEE, 2007, to appear.